Project Name: End-To-End ML with Deployment Car-Price-Prediction (FileUse) – Random Forest - Regression

Table of Contents

Demo

In [71]:
```
df.describe()
```

Out[71]:

	Year	Selling_Price	Present_Price	Kms_Driven	Owner
count	301.000000	301.000000	301.000000	301.000000	301.000000
mean	2013.627907	4.661296	7.628472	36947.205980	0.043189
std	2.891554	5.082812	8.644115	38886.883882	0.247915
min	2003.000000	0.100000	0.320000	500.000000	0.000000
25%	2012.000000	0.900000	1.200000	15000.000000	0.000000
50%	2014.000000	3.600000	6.400000	32000.000000	0.000000
75%	2016.000000	6.000000	9.900000	48767.000000	0.000000
max	2018.000000	35.000000	92.600000	500000.000000	3.000000

In [72]:
```
final_dataset=df[['Year','Selling_Price','Present_Price','Kms_Driven','Fuel_Type','Seller_Type','Transmission','Owner']]
```

In [73]:
```
final_dataset.head()
```

Out[73]:

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0

In [75]:
```
final_dataset.head()
```

Out[75]:

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Current Year
0	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0	2020
1	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0	2020
2	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0	2020
3	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0	2020
4	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0	2020

In [76]:
```
final_dataset['no_year']=final_dataset['Current Year']- final_dataset['Year']
```

In [77]:
```
final_dataset.head()
```

Out[77]:

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Current Year	no_year
0	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0	2020	6
1	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0	2020	7
2	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0	2020	3
3	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0	2020	9
4	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0	2020	6

In [86]:
```
final_dataset.corr()
```

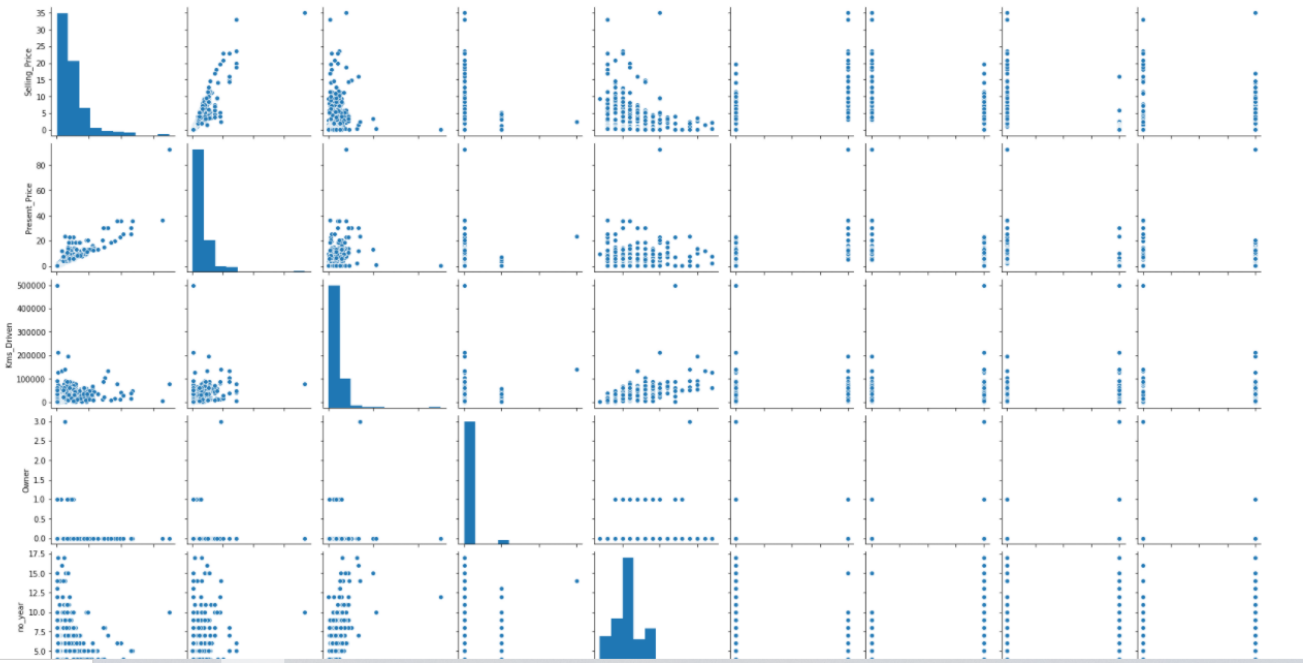
Out[86]:

	Selling_Price	Present_Price	Kms_Driven	Owner	no_year	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Individual	Transmission_Manual
Selling_Price	1.000000	0.878983	0.029187	-0.088344	-0.236141	0.552339	-0.540571	-0.550724	-0.367128
Present_Price	0.878983	1.000000	0.203647	0.008057	0.047584	0.473306	-0.465244	-0.512030	-0.348715
Kms_Driven	0.029187	0.203647	1.000000	0.089216	0.524342	0.172515	-0.172874	-0.101419	-0.162510
Owner	-0.088344	0.008057	0.089216	1.000000	0.182104	-0.053469	0.055687	0.124269	-0.050316
no_year	-0.236141	0.047584	0.524342	0.182104	1.000000	-0.064315	0.059959	0.039896	-0.000394
uel_Type_Diesel	0.552339	0.473306	0.172515	-0.053469	-0.064315	1.000000	-0.979648	-0.350467	-0.098643
uel_Type_Petrol	-0.540571	-0.465244	-0.172874	0.055687	0.059959	-0.979648	1.000000	0.358321	0.091013
Type_Individual	-0.550724	-0.512030	-0.101419	0.124269	0.039896	-0.350467	0.358321	1.000000	0.063240
nission_Manual	-0.367128	-0.348715	-0.162510	-0.050316	-0.000394	-0.098643	0.091013	0.063240	1.000000

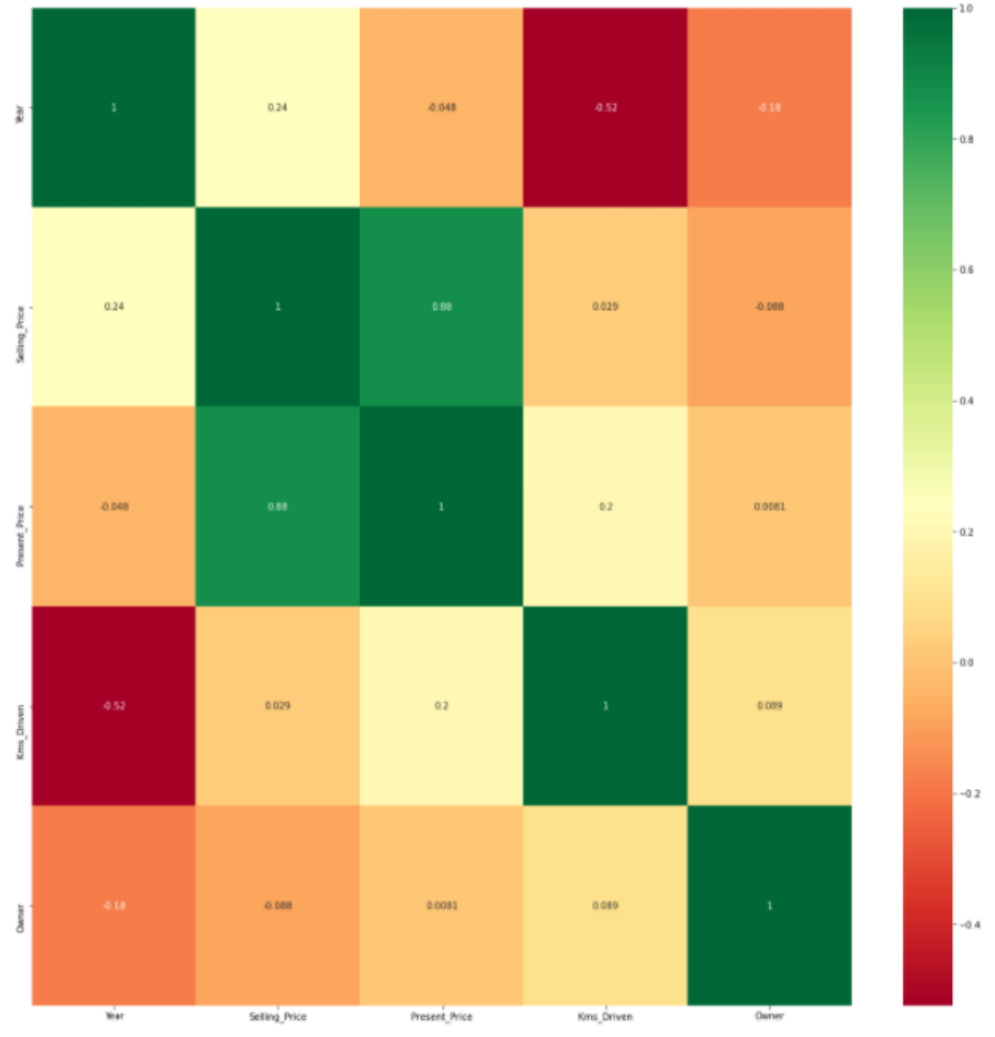
In [87]:
```
import seaborn as sns
```

In [89]:
```
sns.pairplot(final_dataset)
```

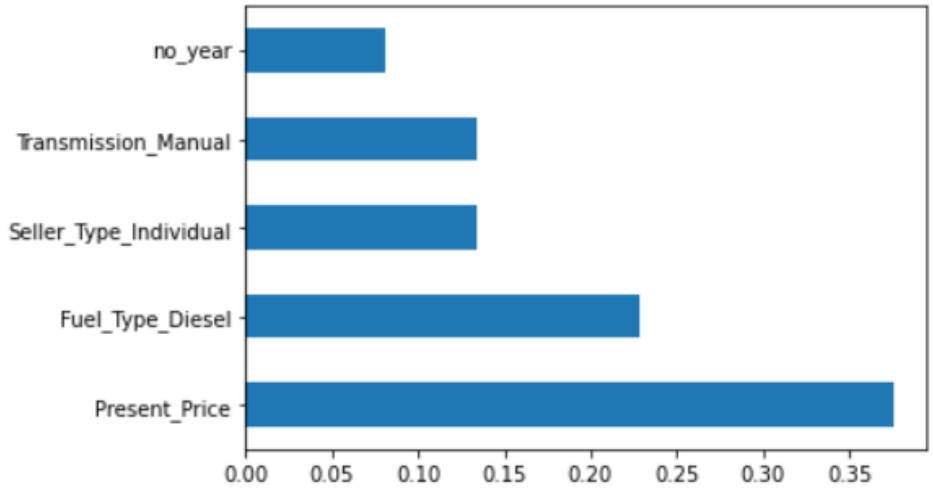
Out[89]:
```
<seaborn.axisgrid.PairGrid at 0x2dbeffe7a88>
```

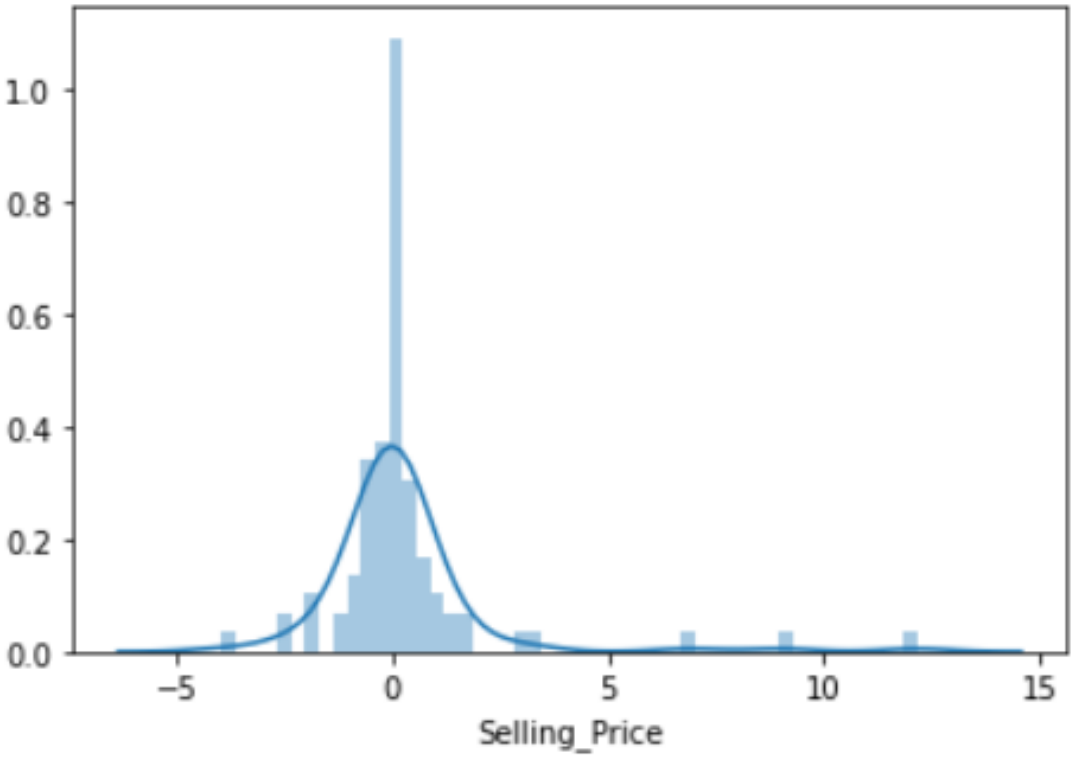



```
import seaborn as sns
#get correlations of each features in dataset
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



In [108]:
```
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(5).plot(kind='barh')
plt.show()
```





In [133]:
```
from sklearn import metrics
```

In [135]:
```
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 0.8849978021977988
MSE: 3.9544237722813156
RMSE: 1.9885733007061408
```

In [136]:
```
import pickle
# open a file, where you ant to store the data
file = open('random_forest_regression_model.pkl', 'wb')

# dump information to that file
pickle.dump(rf_random, file)
```

Live Webapp Demo Link

Deployment on Heroku: http://mlcarpriceprediction.herokuapp.com/

Abstract

The purpose of this report will be to use the Car Data to predict price of a car based on certain factors given.
This can be used to gain insight into how and why final price of car is dependent on its usability.
This can also be used as a model to gain a marketing advantage, by advertisement targeting those who are more likely to retain their old cars, or saving money by not targeting the car bookings that are most likely to purchase.

This is diving into Car Price Prediction through Machine Learning Concept.
End to End Project means that it is step by step process, starts with data collection, EDA, Data Preparation which includes cleaning and transforming then selecting, training and saving ML Models, Cross-validation and Hyper-Parameter Tuning and developing web service then Deployment for end users to use it anytime and anywhere.
This repository contains the code for Car Price Prediction using python's various libraries.
It used pandas, matplotlib, seaborn, sklearn and pickle libraries.
These libraries help to perform individually one particular functionality.
Pandas objects rely heavily on Numpy objects.
Matplotlib is a plotting library.
Seaborn is data visualization library based on matplotlib.
Sklearn has 100 to 200 models.
"Pickling" is the process whereby a Python object hierarchy is converted into a byte stream.
The purpose of creating this repository is to gain insights into Complete ML Project.
These python libraries raised knowledge in discovering these libraries with practical use of it.
It leads to growth in my ML repository.
These above screenshots and video in Video_File Folder will help you to understand flow of output.

Motivation

The reason behind building this is, because till now I have worked on individual concepts so I wanted to combine all things that I have learnt till now and create an End-to-End Project that shows whole life cycle of ML Project. In addition to that, as an employee of a company, I should be able to carry out an entire process own is also essential aspect of it. Building end to end project is gave me wholesome approach to handle given data. Hence, I continue to gain knowledge while practicing the same and spread literary wings in tech-heaven.

Acknowledgement

Dataset Available: https://www.kaggle.com/nehalbirla/vehicle-dataset-from-cardekho?select=car+data.csv

The Data

It displays numbers of rows and columns in a given dataset.
This data has 301 total observations.

```
df.shape
```

```
(301, 9)
```

This data has been cleaned of any null values.

```
# check missing values
df.isnull().sum()
```

```
Car_Name         0
Year             0
Selling_Price    0
Present_Price    0
Kms_Driven       0
Fuel_Type        0
Seller_Type      0
Transmission     0
Owner            0
dtype: int64
```

Here are all the features included in the data set, and a short description of them all.

Column_Name	Description
1. Car_Name	This column should be filled with the name of the car.
2. Year	This column should be filled with the year in which the car was bought.
3. Selling_Price	This column should be filled with the price the owner wants to sell the car at.
4. Present_Price	This is the current ex-showroom price of the car.
5. Kms_Driven	This is the distance completed by the car in km.
6. Fuel_Type	Fuel type of the car.
7. Seller_Type	Defines whether the seller is a dealer or an individual.
8. Transmission	Defines whether the car is manual or automatic.
9. Owner	Defines the number of owners the car has previously had.

```
df.head()
```

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0

Analysis of the Data

Let's start by doing a general analysis of the data as a whole, including all the features the Random Forest algorithm will be using.

- Basic Statistics

```
df.describe()
```

	Year	Selling_Price	Present_Price	Kms_Driven	Owner
count	301.000000	301.000000	301.000000	301.000000	301.000000
mean	2013.627907	4.661296	7.628472	36947.205980	0.043189
std	2.891554	5.082812	8.644115	38886.883882	0.247915
min	2003.000000	0.100000	0.320000	500.000000	0.000000
25%	2012.000000	0.900000	1.200000	15000.000000	0.000000
50%	2014.000000	3.600000	6.400000	32000.000000	0.000000
75%	2016.000000	6.000000	9.900000	48767.000000	0.000000
max	2018.000000	35.000000	92.600000	500000.000000	3.000000

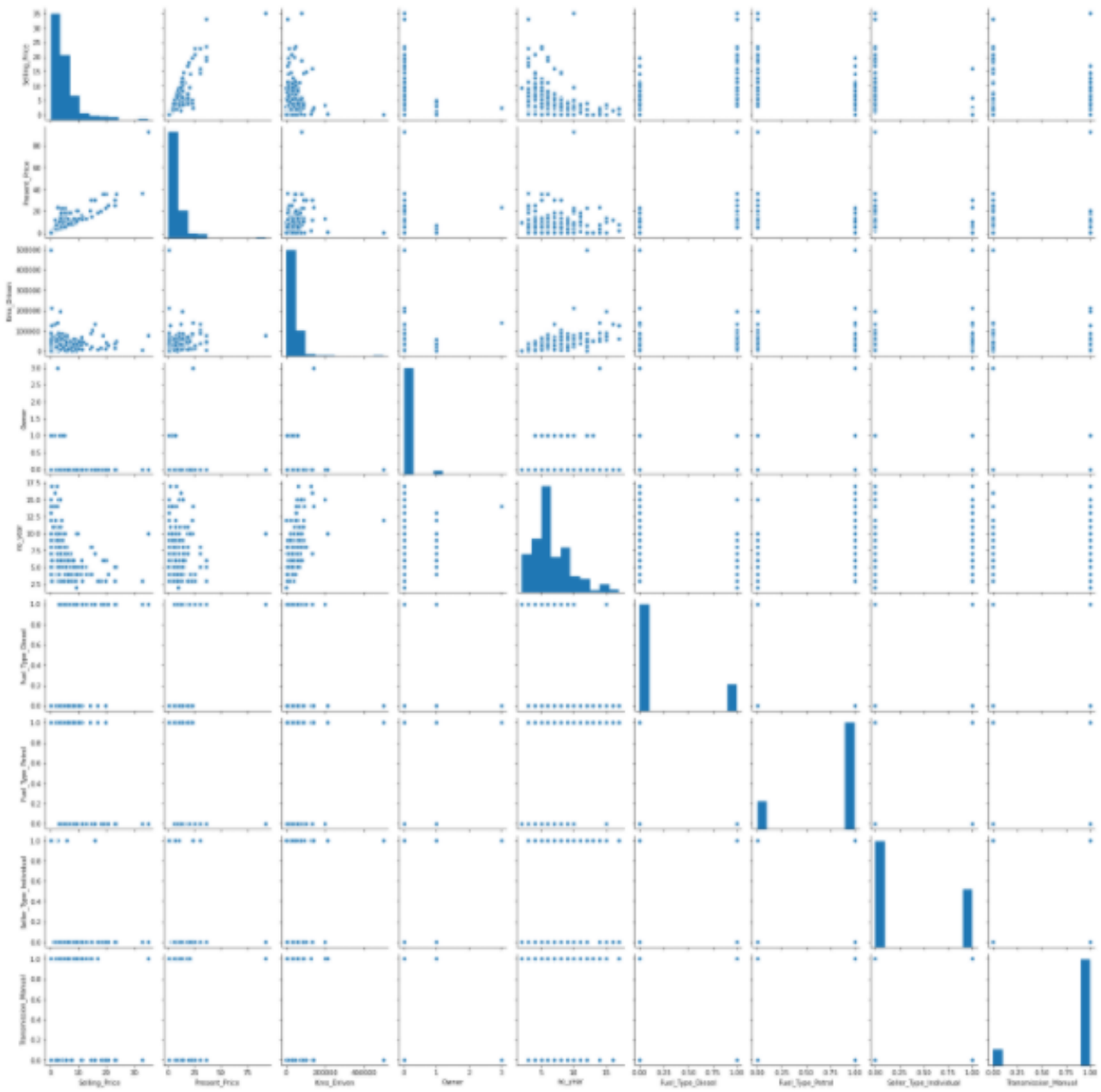
- Graphing of Features

Graph Set 1

```
In [87]: import seaborn as sns
```

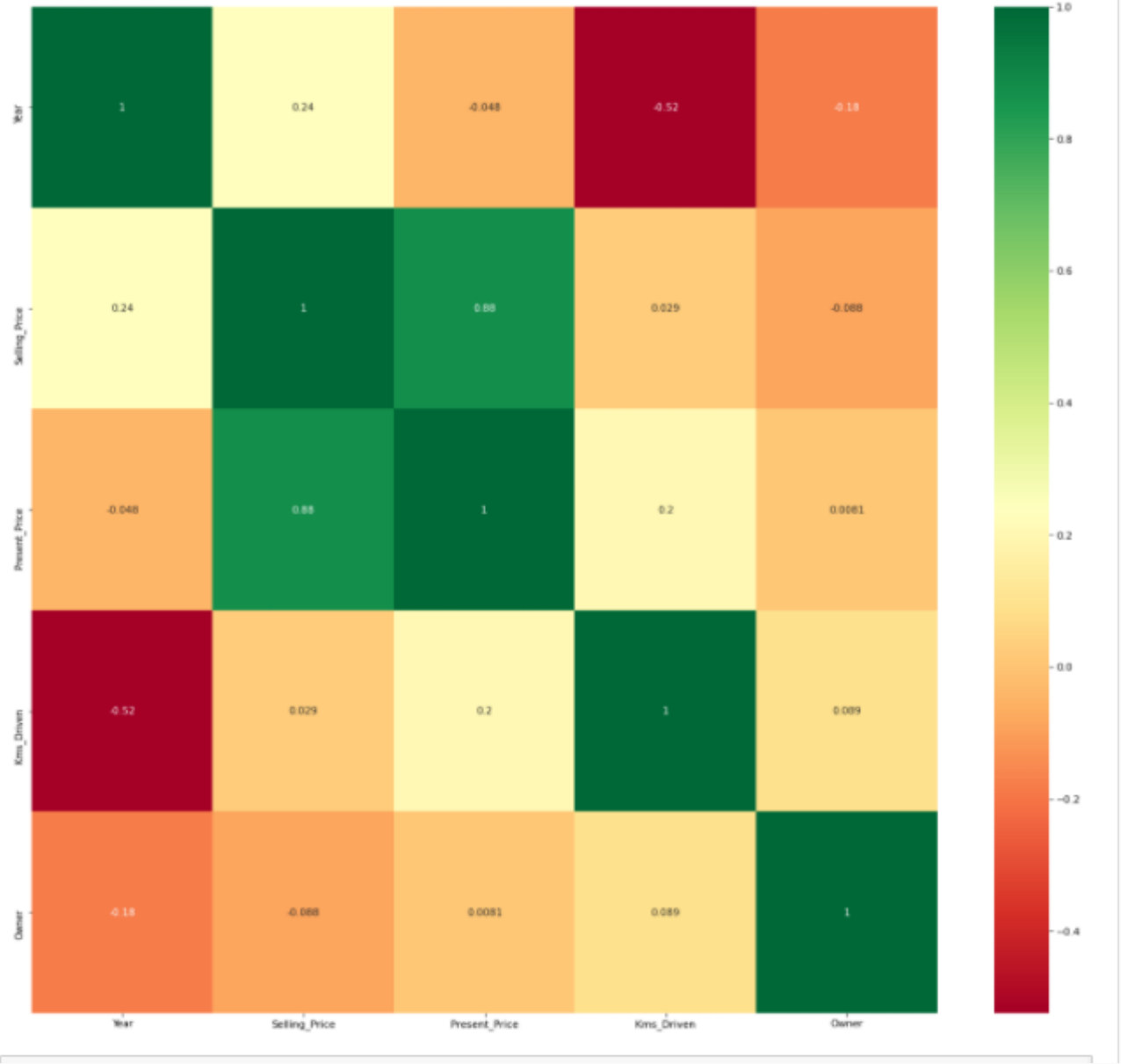
```
In [89]: sns.pairplot(final_dataset)
```

```
Out[89]: <seaborn.axisgrid.PairGrid at 0x2dbeffe7a88>
```



Graph Set 2

```
In [90]:
import seaborn as sns
#get correlations of each features in dataset
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



Modelling

The purpose of these models will be to get effective insight into the following:

1. If price of car depending on various factors:
    - This insight can be used for Market Targeting.
2. Get insight into how changing RMSE of the predictions affect:
    - Spending more money to target the petrol car that are most likely to retain their condition in long run or spending less money for less used car.

Where to Use Random Forest Regression: Random Forest Regression Example

Let's say you want to estimate the average household income in your town. You could easily find an estimate using the Random Forest Algorithm. You would start off by distributing surveys asking people to answer a number of different questions. Depending on how they answered these questions, an estimated household income would be generated for each person.

After you've found the decision trees of multiple people you can apply the Random Forest Algorithm to this data. You would look at the results of each decision tree and use random forest to find an average income between all of the decision trees. Applying this algorithm would provide you with an accurate estimate of the average household income of the people you surveyed.

- Math behind the metrics

Scale-dependent measures (for example: MSE, RMSE, MAE) because it is scale-dependent, i.e. dependent on your dependent variable.
The mean absolute error (MAE) of a model with respect to a test set is the mean of the absolute values of the individual prediction errors on over all instances in the test set.
In statistics, the mean squared error (MSE) or mean squared deviation (MSD) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value. MSE is a risk function, corresponding to the expected value of the squared error loss. The fact that MSE is almost always strictly positive (and not zero) is because of randomness or because the estimator does not account for information that could produce a more accurate estimate.
The MSE is a measure of the quality of an estimator—it is always non-negative, and values closer to zero are better.
When using the Random Forest Algorithm to solve regression problems, you are using the mean squared error (MSE) to how your data branches from each node.
This formula calculates the distance of each node from the predicted actual value, helping to decide which branch is the better decision for your forest. Here, *yi* is the value of the data point you are testing at a certain node and *fi* is the value returned by the decision tree.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (fi - yi)^2$$

Where N is the number of data points,
fi is the value returned by the model and
yi is the actual value for data point i.

RMSE is computed as RMSE = mean((observeds - predicteds)^2) %>% sqrt() . The lower the RMSE, the better the model.
Try to play with other input variables, and compare your RMSE values. The smaller the RMSE value, the better the model. Also, try to compare your RMSE values of both training and testing data. If they are almost similar, your model is good.
The Mean Squared Error (MSE) is a measure of how close a fitted line is to data points.
The MSE has the units squared of whatever is plotted on the vertical axis. Another quantity that we calculate is the Root Mean Squared Error (RMSE). It is just the square root of the mean square error.

The MAE is a linear score which means that all the individual differences are weighted equally in the average. The RMSE is a quadratic scoring rule which measures the average magnitude of the error. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. Therefore, MAE is better than RMSE.

Algorithm for Random Forest Regression:

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

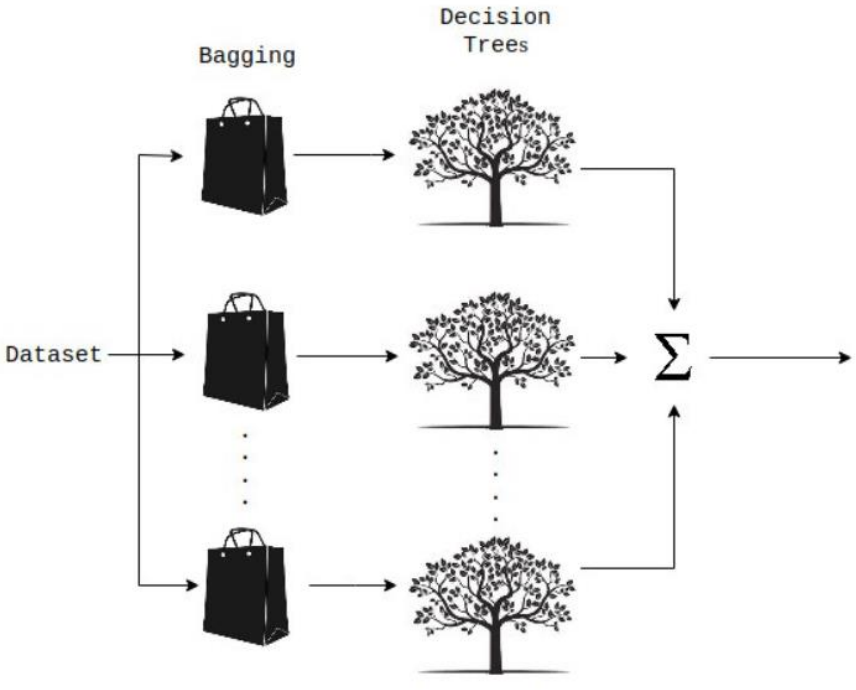
To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

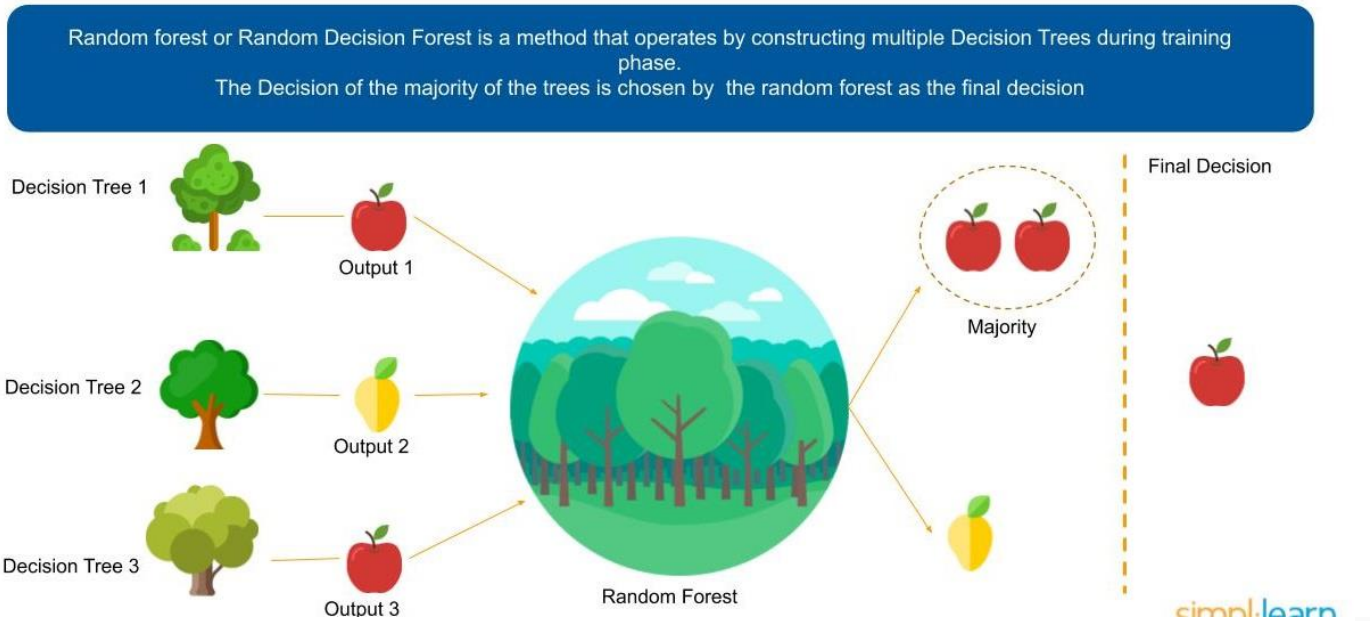
Reference Link: https://deepai.org/machine-learning-glossary-and-terms/random-forest
https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76

- Model Architecture Process Through Visualization

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly known as bagging. Bagging, in the Random Forest method, involves training each decision tree on a different data sample where sampling is done with replacement.



Random Forest Regression: Process



The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

Reference:

When Does RF Regression Fails and Why - https://neptune.ai/blog/random-forest-regression-when-does-it-fail-and-why

I will use RandomizedSearchCV from sklearn to optimize our hyperparamaters.
Now I have plugged these back into the model to see if it improved our performance.

- Quick Notes

Step 1: Imported required libraries.
Step 2: Read Data.
Step 3: Analysed the data and created dummies.
Step 4: Visualized Data.
Step 5: Extracted the Features and applied feature importance by using ensembling technique of ExtraTreesRegression.
Step 6: Plotted Feature Importance.
Step 7: Split the data.
Step 8: Applied Random Forest Regressor technique.
Step 9: Tuned Hyper-parameters through RandomizedSearchCV.
Step 10: Visualized data.
Step 11: Model Evaluation Step - Calculated MAE, MSE, RMSE.
Step 12: Saved the model.
Step 13: Created Web App.

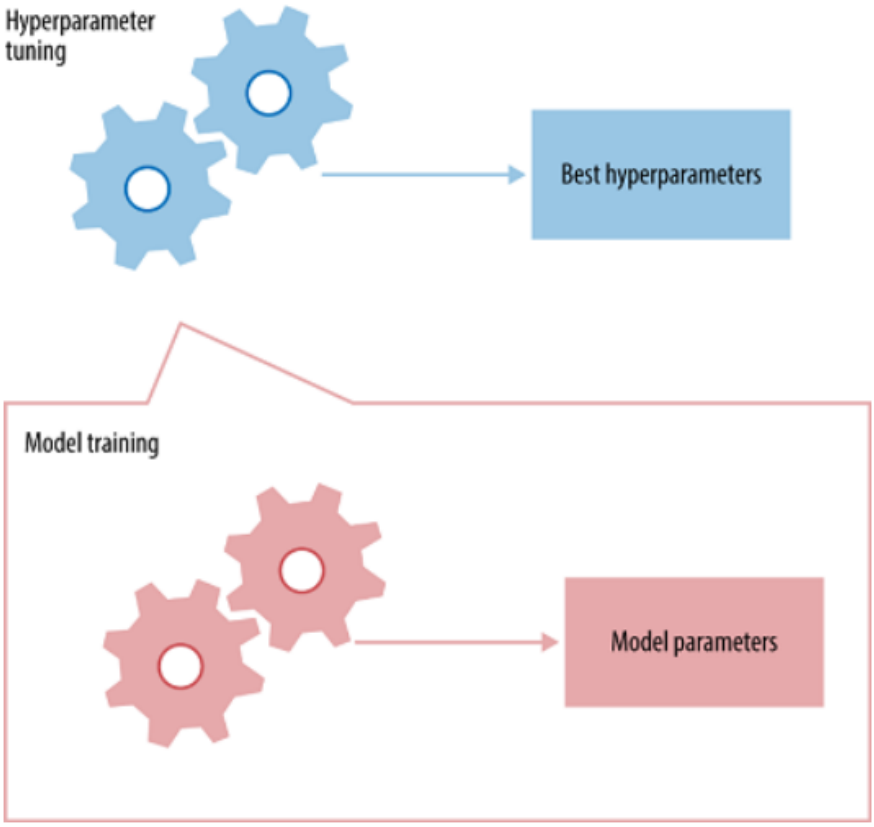
- The Model Analysis

Imported necessary libraries - When we import modules, we're able to call functions that are not built into Python. Some modules are installed as part of Python, and some we will install through pip. Making use of modules allows us to make our programs more robust and powerful as we're leveraging existing code.

Read Data - You can import tabular data from CSV files into pandas data frame by specifying a parameter value for the file.

Analysed the data and created dummies – Analysing data through it shape or describe or missing values to understand behaviour of data. Then, create a dummy variable in Python using Pandas get_dummies method. Specifically, we will generate dummy variables for a categorical variable.

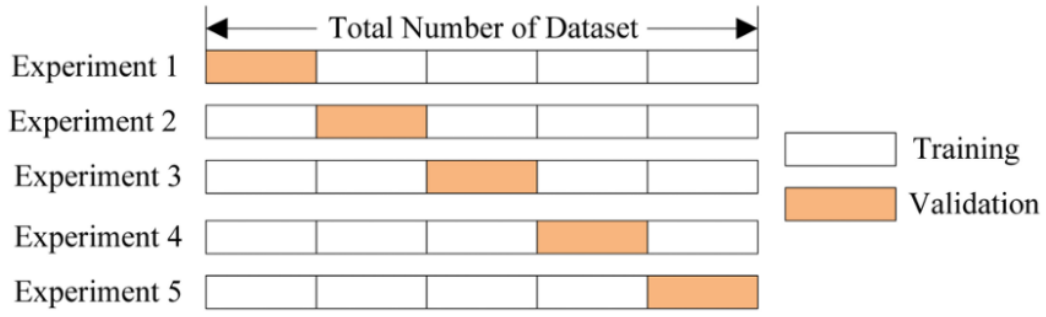
Visualized the data - Visualization gives insights into entire dataset at glance.
Extracted the Features and applied feature importance by using ensembling technique of ExtraTreesRegression - Feature importance refers to techniques that assign a score to input features based on how useful they are at predicting a target variable. Feature importance scores play an important role in a predictive modelling project, including providing insight into the data, insight into the model, and the basis for dimensionality reduction and feature selection that can improve the efficiency and effectiveness of a predictive model on the problem. This can be achieved by using the importance scores to select those features to delete (lowest scores) or those features to keep (highest scores). This is a type of feature selection and can simplify the problem that is being modelled, speed up the modelling process (deleting features is called dimensionality reduction), and in some cases, improve the performance of the model. For example, the results suggest perhaps two or three of the 10 features as being important to prediction.
Plotted Feature Importance – Provides better visualization for decision making in few seconds rather looking at large confusing numbers.
Split the data - All machine learning models require us to provide a training set for the machine so that the model can train from that data to understand the relations between features and can predict for new observations. When we are provided a single huge dataset with too much of observations, it is a good idea to split the dataset into to two, a training_set and a test_set, so that we can test our model after it has been trained with the training_set. Scikit-learn comes with a method called train_test_split to help us with this task.
RF Model Fitting – Training the data.
Hyper-parameter tuning through cross-validation - The best way to think about hyperparameters is like the settings of an algorithm that can be adjusted to optimize performance, just as we might turn the knobs of an AM radio to get a clear signal (or your parents might have!). While model parameters are learned during training — such as the slope and intercept in a linear regression — hyperparameters must be set by the data scientist before training. In the case of a random forest, hyperparameters include the number of decision trees in the forest and the number of features considered by each tree when splitting a node. (The parameters of a random forest are the variables and thresholds used to split each node learned during training). Scikit-Learn implements a set of sensible default hyperparameters for all models, but these are not guaranteed to be optimal for a problem. The best hyperparameters are usually impossible to determine ahead of time, and tuning a model is where machine learning turns from a science into trial-and-error based engineering.



Hyperparameters and Parameters

Hyperparameter tuning relies more on experimental results than theory, and thus the best method to determine the optimal settings is to try many different combinations evaluate the performance of each model. However, evaluating each model only on the training set can lead to one of the most fundamental problems in machine learning: overfitting. If we optimize the model for the training data, then our model will score very well on the training set, but will not be able to generalize to new data, such as in a test set. When a model performs highly on the training set but poorly on the test set, this is known as overfitting, or essentially creating a model that knows the training set very well but cannot be applied to new problems. It's like a student who has memorized the simple problems in the textbook but has no idea how to apply concepts in the messy real world. An overfit model may look impressive on the training set, but will be useless in a real application. Therefore, the standard procedure for hyperparameter optimization accounts for overfitting through cross validation.

The technique of cross validation (CV) is best explained by example using the most common method, K-Fold CV. When we approach a machine learning problem, we make sure to split our data into a training and a testing set. In K-Fold CV, we further split our training set into K number of subsets, called folds. We then iteratively fit the model K times, each time training the data on K-1 of the folds and evaluating on the Kth fold (called the validation data). As an example, consider fitting a model with K = 5. The first iteration we train on the first four folds and evaluate on the fifth. The second time we train on the first, second, third, and fifth fold and evaluate on the fourth. We repeat this procedure 3 more times, each time evaluating on a different fold. At the very end of training, we average the performance on each of the folds to come up with final validation metrics for the model.



5 Fold Cross Validation (Source)

For hyperparameter tuning, we perform many iterations of the entire K-Fold CV process, each time using different model settings. We then compare all of the models, select the best one, train it on the full training set, and then evaluate on the testing set. This sounds like an awfully tedious process! Each time we want to assess a different set of hyperparameters, we have to split our training data into K fold and train and evaluate K times. If we have 10 sets of hyperparameters and are using 5-Fold CV, that represents 50 training loops. Fortunately, as with most problems in machine learning, someone has solved our problem and model tuning with K-Fold CV can be automatically implemented in Scikit-Learn.
Using Scikit-Learn's RandomizedSearchCV method, we can define a grid of hyperparameter ranges, and randomly sample from the grid, performing K-Fold CV with each combination of values.

Refer here for reference: https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

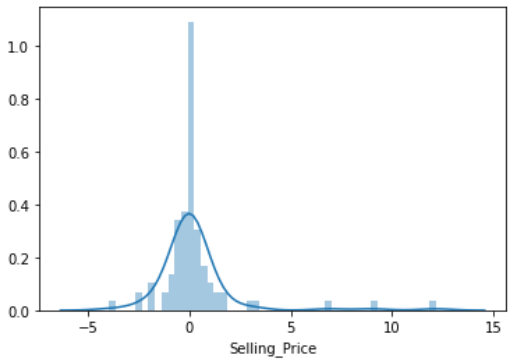
Calculated MAE, MSE, RMSE – To evaluate the prediction error rates and model performance in regression analysis. Once you have obtained your error metric/s, take note of which X's have minimal impacts on y. Removing some of these features may result in an increased accuracy of your model. MAE: The easiest to understand. Represents average error.
MSE: Similar to MAE but noise is exaggerated and larger errors are "punished". It is harder to interpret than MAE as it's not in base units, however, it is generally more popular.
RMSE: Most popular metric, similar to MSE, however, the result is square rooted to make it more interpretable as it's in base units. It is recommended that RMSE be used as the primary metric to interpret your model. All of them require two lists as parameters, with one being your predicted values and the other being the true values.
Saved the model – Saving the created model for future reference and use so that we can directly access it rather than to go through full cycle again.
Created Web App – It is made in flask and for end-users to use it.

Checking the Model Visualization

- Basic Model Evaluation Graphs

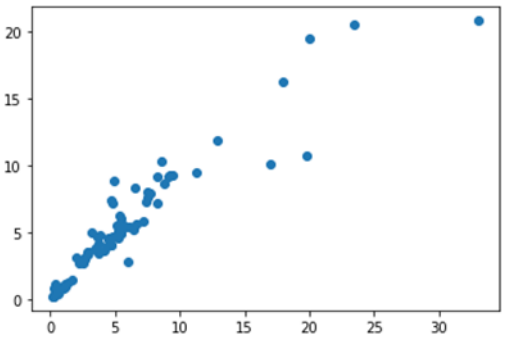
```
sns.distplot(y_test-predictions)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2dbf66939c8>
```




```
plt.scatter(y_test,predictions)
```

```
<matplotlib.collections.PathCollection at 0x2dbf6610708>
```

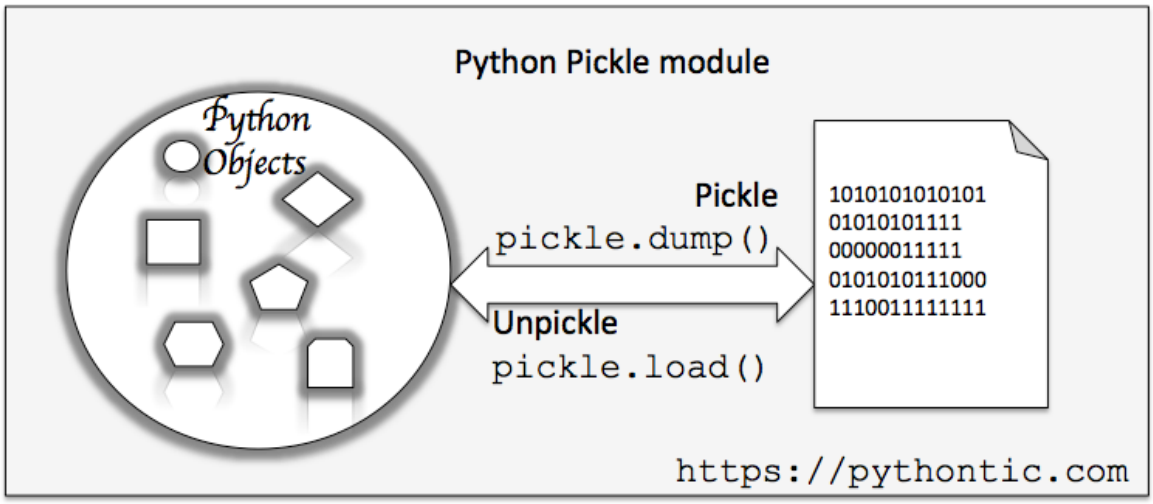


Creation of App

Here, I am creating Flask App. Loading the model that we saved then calling and calculating all the factors that are necessary in deciding price of car.
Displaying message if condition matched and if not matched using loop.
Get the user entered value from the predict class and render respective .html page for solution.
You can create it with Streamlit also.

Technical Aspect

Pandas module mainly works with the tabular data. It contains Data Frame and Series. Pandas is 18 to 20 times slower than Numpy. Pandas is seriously a game changer when it comes to cleaning, transforming, manipulating and analyzing data.
Matplotlib is used for EDA. Visualization of graphs helps to understand data in better way than numbers in table format. Matplotlib is mainly deployed for basic plotting. It consists of bars, pies, lines, scatter plots and so on. Inline command display visualization inline within frontends like in Jupyter Notebook, directly below the code cell that produced it.
Seaborn provides a high-level interface for drawing attractive and informative statistical graphics. It provides a variety of visualization patterns and visualize random distributions.
Sklearn is known as scikit learn. It provides many ML libraries and algorithms for it. It provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.
Pickle in Python is primarily used in serializing and deserializing a Python object structure. In other words, it's the process of converting a Python object into a byte stream to store it in a file/database, maintain program state across sessions, or transport data over the network.



OS module provides a portable way of using operating system dependent functionality.
Need to train_test_split - Using the same dataset for both training and testing leaves room for miscalculations, thus increases the chances of inaccurate predictions.
The train_test_split function allows you to break a dataset with ease while pursuing an ideal model. Also, keep in mind that your model should not be overfitting or underfitting.

Reason for doing Hyper-Parameter Tuning is, Setting the correct combination of hyperparameters is the only way to extract the maximum performance out of models.

The most important arguments in RandomizedSearchCV are n_iter, which controls the number of different combinations to try, and cv which is the number of folds to use for cross validation. RandomizedSearchCV implements a “fit” and a “score” method. It also implements “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used. If at least one parameter is given as a distribution, sampling with replacement is used.
In the realm of machine learning, the random forest regression algorithm can be more suitable for regression problems than other common and popular algorithms. Below are a few cases where you’d likely prefer a random forest algorithm over other regression algorithms:

1. There are non-linear or complex relationships between features and labels.
2. You need a model that’s robust, meaning its dependence on the noise in the training set is limited. The random forest algorithm is more robust than a single decision tree, as it uses a set of uncorrelated decision trees.
3. If your other linear model implementations are suffering from overfitting, you may want to use a random forest.

Extra Trees is an ensemble machine learning algorithm that combines the predictions from many decision trees.
It is related to the widely used random forest algorithm. It can often achieve as-good or better performance than the random forest algorithm, although it uses a simpler algorithm to construct the decision trees used as members of the ensemble.
It is also easy to use given that it has few key hyperparameters and sensible heuristics for configuring these hyperparameters.
The Extra Trees algorithm works by creating a large number of unpruned decision trees from the training dataset. Predictions are made by averaging the prediction of the decision trees in the case of regression or using majority voting in the case of classification.

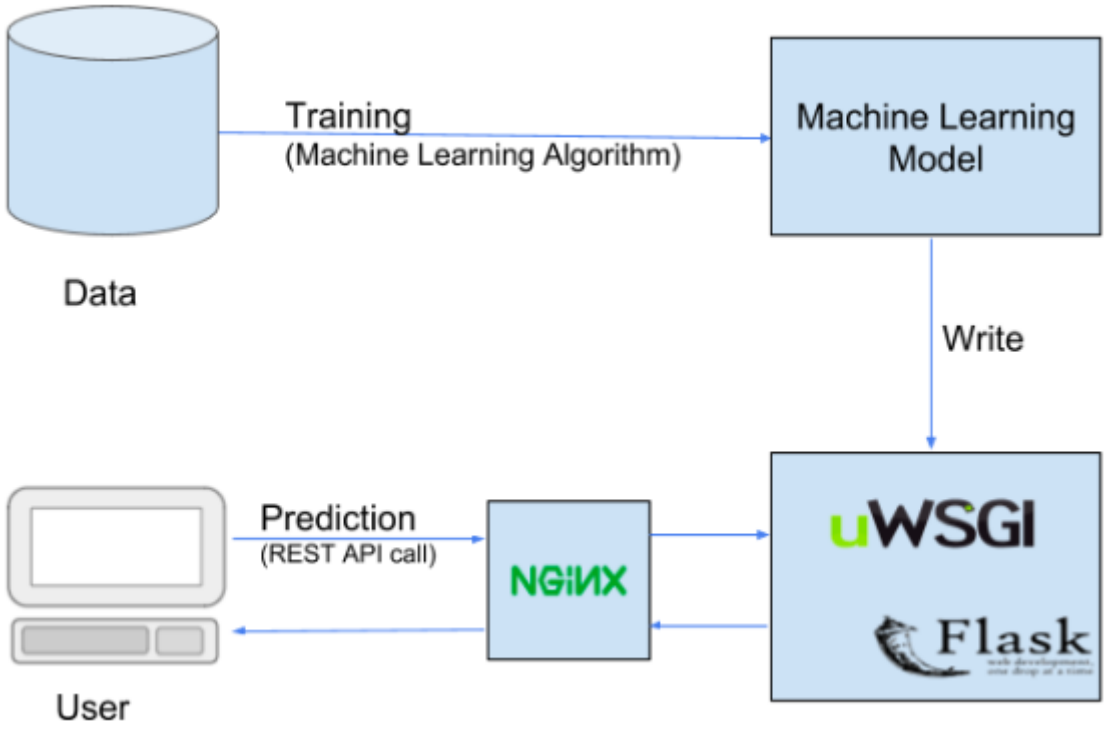
- Regression: Predictions made by averaging predictions from decision trees.

- Classification: Predictions made by majority voting from decision trees.

Unlike bagging and random forest that develop each decision tree from a bootstrap sample of the training dataset, the Extra Trees algorithm fits each decision tree on the whole training dataset. Like random forest, the Extra Trees algorithm will randomly sample the features at each split point of a decision tree. Unlike random forest, which uses a greedy algorithm to select an optimal split point, the Extra Trees algorithm selects a split point at random.

There are three main hyperparameters to tune in the algorithm; they are the number of decision trees in the ensemble, the number of input features to randomly select and consider for each split point, and the minimum number of samples required in a node to create a new split point.

Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.



Installation

Using intel core i5 9th generation with NVIDIA GFORECE GTX1650.
Windows 10 Environment Used.
Already Installed Anaconda Navigator for Python 3.x
The Code is written in Python 3.8.
If you don't have Python installed you can install Python from its official site.
If you are using a lower version of Python you can upgrade using the pip package, ensuring you have the latest version of pip, *python* -m pip install --*upgrade pip and press Enter*.

Run/How to Use/Steps

Keep your internet connection on while running or accessing files and throughout too.
Follow this when you want to perform from scratch.
Open Anaconda Prompt, Perform the following steps:

cd <PATH>
pip install pandas
pip install matplotlib
pip install seaborn
pip install sklearn
pip install requests
pip install scipy
pip install urllib3
pip install jsonify
Note: If it shows error as 'No Module Found' , then install relevant module.
You can also create requirement.txt file as, pip freeze > requirements.txt
Create Virtual Environment:
conda create -n ee1 python=3.7
y
conda activate ee1
cd <PATH-TO-FOLDER>
run .py or .ipynb files.
Paste URL to browser to check whether working locally or not.

Follow this when you want to just perform on local machine.
Download ZIP File.
Right-Click on ZIP file in download section and select Extract file option, which will unzip file.
Move unzip folder to desired folder/location be it D drive or desktop etc.
Open Anaconda Prompt, write cd <PATH> and press Enter.
eg: cd C:\Users\Monica\Desktop\Projects\Python Projects 1\
23)End_To_End_Projects\Project_1_ML_FileUse_EndToEnd_CarPricePrediction>
Project_ML_CarPricePrediction

conda create -n ee1 python=3.7
y
conda activate ee1
In Anconda Prompt, pip install -r requirements.txt to install all packages.
In Anconda Prompt, write python app.py and press Enter.
Paste URL to browser to check whether working locally or not.
Please be careful with spellings or numbers while typing filename and easier is just copy filename and then run it to avoid any silly errors.
Note: cd <PATH>
[Go to Folder where file is. Select the path from top and right-click and select copy option and paste it next to cd one space <path> and press enter, then you can access all files of that folder]
[cd means change directory]

Directory Tree/Structure of Project

Folder: 23)End_To_End_Projects > Project_1_ML_FileUse_EndToEnd_CarPricePrediction > Project_ML_CarPricePrediction

Model_Building.ipynb

app.py

car data.csv

Procfile

random_forest_regression_model.pkl

requirements.txt

Folder: 23)End_To_End_Projects > Project_1_ML_FileUse_EndToEnd_CarPricePrediction > Project_ML_CarPricePrediction >templates

index.html

```
C:.
└───Project_ML_CarPricePrediction
    │   app.py
    │   car data.csv
    │   Model_Building.ipynb
    │   Procfile
    │   random_forest_regression_model.pkl
    │   requirements.txt
    │
    ├───Images_packages used
    │       11.png
    │       12.jpeg
    │       13.jpg
    │       15.png
    │       16.jpeg
    │
    ├───Images_screenshot
    │       1.png
    │       10.png
    │       11.png
    │       12.png
    │       13.png
    │       14.png
    │       15.png
    │       16.png
    │       17.png
    │       18.png
    │       19.jpg
    │       2.png
    │       20.png
    │       21.png
    │       22.png
    │       23.png
    │       24.png
    │       25.png
    │       26.png
    │       3.png
    │       4.png
    │       5.png
    │       6.png
    │       7.png
    │       8.png
    │       9.png
    │
    ├───templates
    │       index.html
    │
    └───Video_File
            Car_Price_Prediction.mp4
```

To Do/Future Scope

Can try with another dataset.

Can try other techniques of data pre-processing.

Can deploy on AWS and Google Cloud.

Technologies Used/System Requirements/Tech Stack

Download the Material

You can Download Dataset from here: https://github.com/monicadesAI-tech/Project_65/blob/main/car%20data.csv
You can Download Entire Project from here: https://github.com/monicadesAI-tech/Project_65
You can Download Model Building from here: https://github.com/monicadesAI-tech/Project_65/blob/main/Model_Building.ipynb
Download Saved Model from here: https://github.com/monicadesAI-tech/Project_65/blob/main/random_forest_regression_model.pkl
You can Download Web App_File from here: https://github.com/monicadesAI-tech/Project_65/blob/main/app.py
Download Dependencies from here: https://github.com/monicadesAI-tech/Project_65/blob/main/requirements.txt
You can see website here: https://github.com/monicadesAI-tech.github.io/project/carprice.html

Conclusion

- Modelling

Using XGBoost along with RF and creating their stack, may give better results with less pre-processing.

- Analysis

RMSE is 1.98 which is pretty good however can work more on MSE by above mentioned approach.

Credits

Krish Naik Channel

Paper Citation

https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf